

Travail d'Etude et de Recherches

Master 1 STIC 2004/2005

Cahier des Charges

Plugin Eclipse pour la composition de préoccupations dans le langage Java

Développement de JAdapt 2



Réalisé par :

Bonfils David *Julien Nicolas*

Férut Térence *Leroy Sébastien*

Sous la tutelle de messieurs :

Lahire Philippe et

Crescenzo Pierre

Plan

- I) Introduction
  - Résumé - Objectifs
  - Fournitures
  - Définitions et Acronymes
- II) Organisation du projet
  - Processus
    - Phase d'étude et d'analyse
    - Phase de développement
  - Organisation structurelle
  - Limites
- III) Gestion
  - Objectifs et priorités
  - Hypothèses, dépendances et contraintes.
  - Gestion du risque
  - Moyens de contrôle
- IV) Technique
  - Méthodes et outils employés
  - Documentation
- V) Calendrier
- VI) Fonctionnalités
- VII) Contraintes non-fonctionnelles
- VIII) Contacts

I) Introduction

Résumé - Objectifs

Dans le cadre de sa thèse de doctorat à l'Université de Nice-Sophia Antipolis entre 2002 et 2004, Laurent Quintian a proposé un modèle très ambitieux, proposant une amélioration de la réutilisation des préoccupations dans les langages orientés objets.

Ce modèle, implémenté sous Java avec succès sous le nom de JAdapt, propose les avantages de la programmation orientée aspects et de la programmation par sujets.

Nonobstant, la version livrée par M. Quintian reste difficile à aborder (le code est difficilement lisible et mal structuré), et à utiliser et peut encore être améliorée de façon notable sur certains points. De plus elle n'est plus compatible avec la dernière version d'Eclipse.

Nous nous proposons donc de reprendre son travail, de porter son plugin sur Eclipse 3, d'y intégrer le nouveau modèle plus élaboré proposé par Philippe Lahire, et de revoir la structure générale de l'application pour rendre le code plus extensible et plus simple afin d'améliorer son intégration dans Eclipse. Mais de l'avis de M. Lahire, il faudra d'abord se focaliser sur une implémentation claire et propre de notre travail, quitte à ce qu'elle soit incomplète, dans l'optique où elle sera reprise dans le futur par d'autres personnes (aucune maintenance de notre part de prévu).

Fournitures

L'actuelle version de JAdapt est un plugin pour Eclipse 2.X. Nous rendrons donc un nouvelle version de JAdapt, toujours sous forme de plugin mais pour Eclipse 3, accompagné d'un manuel d'utilisation en PDF et d'une documentation technique au format HTML.

Définitions et Acronymes

Préoccupation : Ou *Concern* en anglais, propriété non-fonctionnelle d'une application telle que la persistance, la distribution, le traçage ou encore le temps-réel.

Réification : Art de transformer quelque chose d'abstrait en chose. En POO il s'agit de créer des Objets correspondant à des notions abstraites.

EMF : EMF, *Eclipse Modeling Framework*, est un outil proposant un méta-modèle permettant d'utiliser des modèles XMI, un éditeur de ce modèle ainsi qu'une fonctionnalité de génération de code en Java.

RCP : RCP, *Rich Client Platform*

POO : Programmation Orientée Objets - paradigme servent de support en général à la POA et à la POS.

POA : Programmation Orientée Aspects, paradigme de programmation visant à réduire à néant l'entrelacement de code entre différentes propriétés fonctionnelles et non-fonctionnelle de l'application.

POS : Programmation Orientée Sujets, extension du paradigme Objet permettant la décomposition du code en *Sujets*. L'application finale est composition par *intégration* des sujets.

II) Organisation du projet

Processus

Le processus de développement de notre TER se fera en deux parties disjointes.

Phase d'étude et d'analyse

La première partie se terminera lorsque nous pourrons travailler à plein temps sur le TER. Cette étape est décomposée en plusieurs points :

- Immersion dans le travail de Laurent Quintian. Lecture de sa thèse, de l'article de M. Lahire, étude de son code ...
- Proposition d'une nouvelle architecture pour le moteur de composition des adaptations.
- Etude des outils disponibles (Eclipse ou autres) pour créer un éditeur permettant au développeur de construire facilement un module de composition.
- Familiarisation avec le concept de plugin sous Eclipse.

Phase de développement

Si M. Lahire approuve nos propositions, le développement pourra alors commencer. Notre travail repose sur une version déjà existante et fonctionnant (mais pour une version antérieure d'Eclipse). Nous travaillerons donc de manière **incrémentale**, en n'ajoutant de nouvelles fonctionnalités que si aucune fonctionnalité en développement ne dépend d'elles. De plus, nous ne passerons à la fonctionnalité suivante que lorsque nous jugerons finale la fonctionnalité en cours.

De cette façon nous nous assurons qu'à la fin du TER le produit que nous livrerons sera en principe conforme à nos objectifs, qui je le rappelle, sorti des quelques impératifs est avant tout de produire un code de qualité même s'il n'est totalement fini.

Nous réutiliserons les exemples fournis par Laurent Quintian afin de nous assurer que notre implémentation reste compatible à la sienne, modulo près ses modules de compositions qui seront désormais entrés dans un éditeur. Nous pourrons rendre totale la compatibilité si notre plugin pouvait toujours accepter les modules aux formats XML. Nous essaierons également de développer plusieurs exemples de notre cru, qui nous serviront de tests.

Organisation structurelle

Les tâches étant assez bien découplées les unes des autres, nous pouvons donc limiter au maximum les inter-dépendances entre le travail de chacun et *de facto* accroître notre rendement avec une bonne organisation.

Nous prevoyons donc de nous séparer en deux équipes de deux personnes. La première travaillera sur la modification du code de JAdapt pour y intégrer les modifications ennoncées tantôt. La seconde équipe s'occupera de la création de l'éditeur et de l'intégration de notre plugin dans Eclipse.

La modification du code s'étendra probablement jusqu'au dernier moment car nous prévoyons de reprendre le plus en profondeur possible l'architecture de l'application et ses fonctionnalités. Il n'est donc pas impossible que les effectifs des équipes évoluent dans le temps, suivant l'avancée du travail de chacun, et de l'accomplissement des objectifs jugés impératifs.

Limites

Les limites de notre plugin seront avant tout les limites de l'environnement dans lequel nous nous inscrivons, à savoir Eclipse. Nous assurerons sa compatibilité avec Eclipse 3 et ses outils actuels, mais rien ne nous garantis que ces derniers ne vont pas évolués eux aussi dans le temps, et à notre détriment. D'autre part nous espérons améliorer au maximum le pouvoir d'expressivité de notre plugins grâce au nouveau modèle de M. Lahire mais nous savons que cette expressivité risque de ne pas égaler celles de langages (ou extension de langages) purement orientés aspects ou sujets.

III) Gestion

Objectifs et priorités

Nous réaliserons un portage de JAdapt d'Eclipse 2 à Eclipse 3 avec une refonte profonde de l'architecture de l'application dans le but de rendre cette dernière la plus extensible possible. Nous ne réaliserons probablement pas de maintenance de notre code *a posteriori*, il faut donc que ce dernier soit très clair et lisible, même s'il est incomplet. De plus nous proposerons un éditeur permettant au développeur de créer facilement son module de composition et ainsi d'utiliser avec aisance notre plugin.

Hypothèses, dépendances et contraintes.

Pour commencer nous devons supposer qu'Eclipse est un environnement stable et qu'il n'existe aucun bogue lié à la création ou l'utilisation de plugin (et en particulier du nôtre). Nous sommes effectivement très dépendant de la plate-forme Eclipse, et comme nous tendrons à incorporer notre version de JAdapt au maximum dans Eclipse 3 cette dépendance ne pourra que s'accroître.

Enfin nous n'oublierons pas que notre plugin ne sera pas utilisé si son utilisation est délicate ou malaisée ; nous rechercherons donc constamment *l'intuitivité* et *l'ergonomie* optimales.

Gestion du risque

Etant donné que les tâches ne s’effectueront pas séquentiellement (ou très peu) mais plutôt parallèlement le risque est minime.

Tout d'abord nous parallélisons les trois tâches fondamentales (création de l'éditeur, refonte de l'architecture du moteur, intégration du plugin) que nous traiterons dès le début de la phase de développement. Ainsi nous pensons que dans un délai convenable (*cf* Calendrier) nous pouvons avoir accompli nos objectifs prioritaires. Ensuite, de manière séquentielle et incrémentale nous tenterons de développer un maximum de fonctionnalités optionnelles. De plus nous avons aménagé notre Calendrier de sorte que si une des tâches prioritaires devait prendre plus de temps que prévu, nous puissions justement empiéter sans risque sur le temps que nous allouons aux fonctionnalités optionnelles.

Enfin pour ne pas risquer de perdre du temps inutilement sur une tâche nous organiserons très régulièrement des réunions dont l'objectif sera pour chacun de présenter ses travaux, ses difficultés et ses prévisions. Ainsi si un de nous a un problème, il saura que toute l'équipe pourra l'aider et le conseiller. Le deuxième objectif de ces réunions sera aussi de **valider** les tâches terminées. Enfin nous nous assurons ainsi que toute l'équipe sera constamment au courant de tout ce qui se passe, bien que les tâches soient pourtant clairement découplées.

Nous valorisons ainsi le travail d'équipe et l'initiative en laissant à chacun la possibilité d'apporter un point de vue personnel ou une solution aux problèmes de ses collègues.

En clair le mot d'ordre est la **communication**.

Moyens de contrôle

Nous profitons déjà des exemples et des codes de tests de Laurent Quintian. Un de nos objectifs étant de conservé une compatibilité parfaite (modulo le module de composition, du moins au début) avec sa première version de JAdapt, ses programmes de tests doivent donc aussi fonctionner avec notre version (avec peut-être le module de composition à refaire, dans le pire des cas).

Nous développerons également nos propres programmes de tests permettant de tester les ajouts de M. Lahire concernant le modèle original.

De plus nous nous efforcerons à rédiger tout au long du développement la documentation de toutes nos méthodes ainsi que celles de Laurent Quintian, souvent mal ou peu documentées. Une bonne documentation réalisée au fur et à mesure permet souvent de corriger moult bogues.

Enfin un des autres objectifs des réunions régulières de l'équipe sera de **valider** une tâche. Tant qu'il demeure un problème nous resterons dessus, nous réfléchirons ensemble pour y apporter une solution, quitte à modifier notre organisation si nécessaire, mais hors de question de rendre un travail approximatif. Cependant une tâche validée par l'équipe entière est considérée comme achevée, et une tâche ne sera considérée comme achevée **que** si elle a été validée par le restant de l'équipe.

IV) Technique

Méthodes et outils employés

- Tout comme son prédécesseur, JAdapt version 2 sera entièrement programmé en Java 1.5 mais utilisera à présent des modules de composition créés à partir d'une application générée par RCP, puis réifiés en Java (alors que JAdapt 1 utilisait un module de composition entré en XML).

- Notre application sera un plugin Eclipse - il va donc de soi qu'Eclipse sera notre plate-forme de développement principale.

- De plus nous essaierons tant que faire se peut d'utiliser des outils d'Eclipse (libres de droits) plutôt que de chercher ailleurs.

Documentation

La documentation se situe au coeur de notre développement. Rappelons que l'un de nos impératifs est de livrer une application **extensible**, il serait inconcevable qu'elle soit mal documentée. Nous commenterons donc toutes les sources existantes (peu commentées jusqu'à présent), les rajouts et modifications que nous ferons, et nous générerons une documentation HTML à l'aide de la Javadoc (disponible en ligne à partir de notre page de suivi dès la fin du développement).
Cette documentation sera accompagnée d'un rapport expliquant les différences entre la première version de JAdapt et la nouvelle version, ainsi qu'un manuel d'utilisation, au format PDF expliquant au développeur comment à la fois installer notre plugin et l'utiliser. Ces documentations seront rédigées à la fois en **français** et en **anglais**, ou simplement en anglais afin d'internationaliser notre plugin et faciliter son intégration dans la communauté des développeurs sous Eclipse.

V) Calendrier

Voir annexe. On distingue bien la partie d'analyse et de recherches (semaines < 18) et la répartition des tâches pour la partie de développement (semaines 18 à 23), par personne ou par équipe. Les principales réunions de validations apparaissent à la fin des semaines 20 et 22.

Notre gestion du risque apparaît clairement sur le planning, on distingue bien le *découplage* des objectifs prioritaires, traités parallèlement dès le début de la phase de développement et normalement terminées dès la fin de la semaine 20. En gris apparaissent les tâches possibles d'être annulées si nous constatons un retard en aval.

VI) Fonctionnalités

- **Réaliser un portage du plugin d'Eclipse 2.X vers 3.X (impératif)** : si nous voulons proposer notre plugin à la communauté d'Eclipse il faut absolument qu'il soit compatible avec la dernière version d'Eclipse.

- **Réaliser un produit extensible (impératif)** : nous n'envisageons pas produire un produit parfait mais puisqu'il ne le sera pas nous devons faire en sorte que quelqu'un puisse l'améliorer s'il le souhaite.
- **Réaliser un éditeur de composition (impératif)** : Entrer le module de composition à la main comme dans JAdapt 1 est un frein au développement de JAdapt, il est indispensable que le développeur puisse entrer facilement et sans connaissances prérequises dans un autre langage un module de composition, même complexe.
- **Revoir l'architecture du moteur de réalisation des adaptations (impératif)** : Le moteur de réalisation des compositions est un composant qui doit pouvoir évoluer dans le futur. Hors sa structure actuelle est un frein à son amélioration. Nous devons donc proposer puis implémenter une nouvelle architecture pour ce moteur.
- **Rajouter des adaptations (optionnel)** : On peut imaginer de nouvelles adaptations non encore implémentées. Si nous prouvons qu'elles offrent un plus réel et que nous réussissons à les implémenter ce sera un plus.
- **Réaliser une vue textuelle dans notre éditeur (optionnel)** : nous essaierons de proposer au développeur une vue textuelle de son adaptation dans un langage type Java-étendu.
- **Compatibilité avec JAdapt 1 (optionnel)** : nous essaierons de faire en sorte que le produit que nous livrerons sera compatible avec la première version de JAdapt. La compatibilité se jouera au niveau du module de composition.
- **Utiliser la compilation incrémentale d'Eclipse (optionnel)** : Eclipse permet de réaliser des compilations "intelligentes", en ne recompilant à chaque fois que ce qui est nécessaire. Hors Laurent Quintian n'a pas su intégrer dans son plugin l'utilisation de ce type de compilation, ses projets sont donc recompilés intégralement à chaque fois. Si c'est à notre portée nous tenterons de réaliser cet ajout fort utile.

VII) Contraintes non-fonctionnelles

Si nous n'arrivons pas à utiliser la compilation incrémentale d'Eclipse, le temps de compilation risque d'être une forte contrainte. En effet si on travaille sur d'énormes projets, tout recompiler peut parfois prendre un temps phénoménal, variant fortement suivant les machines. Il n'en demeure pas moins comme nous le savons tous, que *le temps c'est de l'argent* surtout dans le domaine du développement.

VIII) Contacts

David Bonfils : BonfilsD@echo.unice.fr
Térence Férut : FerutT@echo.unice.fr
Nicolas Julien : JulienN@echo.unice.fr
Sébastien Leroy : LeroyS@echo.unice.fr
Philippe Lahire : Philippe.Lahire@unice.fr
Pierre Crescenzo : Pierre.Crescenzo@unice.fr